泰坦尼克问题的SVM求解

刘浚嘉

上海交通大学机械与动力工程学院

118020910046，[junjialiu@sjtu.edu.cn](mailto:junjialiu@sjtu.edu.cn)

**摘要：**本文利用支持向量机(SVM)方法解决Kaggle泰坦尼克幸存预测问题。其中，详细描述了该问题的凸二次规划模型的建立，应用拉格朗日乘子法以及对偶性原理将其转换为对偶问题，最后应用KKT条件求最优解，得到最优的分类超平面方程。本文使用python语言编写求解程序，首先使用sklearn库中的svm函数求解，之后又基于SVM的原理，重新实现了函数功能并进行求解。

## 一、题目

泰坦尼克幸存预测问题是Kaggle（一个数据建模和数据分析竞赛平台）入门级比赛项目，也是机器学习的基础问题，问题描述如下：

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

题目给出了泰坦尼克号的乘员数据，以及他们的身份标签。根据这些数据，尝试找出乘员存亡与其身份之间的关系。在学习到的模型基础上，预测一名乘客能否幸免于泰坦尼克沉没。

## 二、模型建立

支持向量机，因其英文名为Support Vector Machine，故一般简称SVM，通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

### 2.1 线性分类器

本题是典型的二分类问题，即给定一些数据点，它们分别属于两个不同的类，现在要找到一个线性分类器把这些数据分成两类。如果用x表示数据点，用y表示类别（y可以取1或者-1，分别代表两个不同的类），一个线性分类器的学习目标便是要在n维的数据空间中找到一个超平面（hyper plane）将数据点分为两类，这个超平面的方程可以表示为（中的T代表转置）：

下面举个简单的例子。如下图1所示，现在有一个二维平面，平面上有两种不同的数据，分别用圈和叉表示。由于这些数据是线性可分的，所以可以用一条直线将这两类数据分开，这条直线就相当于一个超平面，超平面一边的数据点所对应的y全是-1 ，另一边所对应的y全是1。

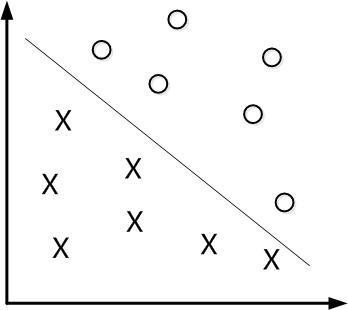


图1 简单线性分类

这个超平面可以用分类函数表示，当等于0的时候，便是位于超平面上的点，而大于0的点对应 的数据点，小于0的点对应的点，如下图2所示：

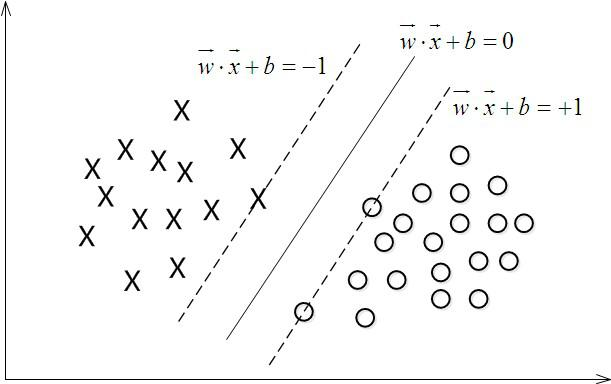


图2 超平面与数据点之间的关系

线性分类问题的关键在于确定超平面的函数表达式。直观上讲，这个超平面应该是最适合分开两类数据的那一个。图3中列出了两种可能的超平面选择，其中黄色的色带表示超平面到训练样本的最短距离，可以理解为学习到的模型的容错性，很明显图3(b)具有更大的最短距离，因此这个模型相较于图3(a)有更好的鲁棒性。因此，判定“最适合”的标准就是超平面到两侧数据点的距离和最大，这就成为了一个最优化问题，优化的目标是找到使间隔最大的那个超平面。

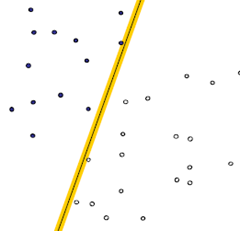


图3(a) 第一种超平面

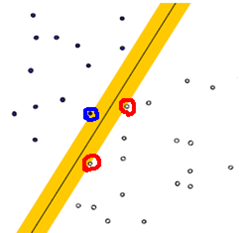


图3(b) 第二种超平面

### 2.2 函数间隙和几何间隙

一个点距离分离超平面的远近可以表示分类预测的确信程度。在超平面已经确定的前提下，能够相对地表示点距离超平面的远近，通过观察的符号与类标记y的符号是否一致可判断分类是否正确，所以，可以用来表示分类的正确性及确信度，即为函数间隙，下面给出其定义：

**定义1（函数间隙）** 对于给定的训练数据集和超平面，定义超平面关于样本点的函数间隙为

定义超平面关于整个训练集的函数间隔为超平面关于中所有样本点的函数间隔的最小值，即

对函数间隔规范化，得到几何间隔，记作。

**定义2（几何距离）** 对于给定的训练数据集和超平面，定义超平面关于样本点的几何间隔为

其中，是向量的范数（norm）。定义超平面关于整个训练集的函数间隔为超平面关于中所有样本点的函数间隔的最小值，即

支持向量机学习的基本思想就是求解能够正确划分训练集并且几何间隔最大的分离超平面（本文主要叙述硬间隔最大化，至于软间隔最大化无非是引入了松弛变量和惩罚项，使模型容错性更强；对于非线性支持向量机所使用的核函数，其原理是对原空间数据进行非线性变换，将其映射到一个希尔伯特空间，使分类的超曲面对应于中的超平面，本文也不再赘述）。

### 2.3 模型建立

该问题可以表示为如下的最优化问题：

根据函数间隔与几何间隔的关系，可以将其表示为函数间隔形式

函数间隔仅是一个常数，其取值并不影响最优化问题的解，此处取1。

将，由于求的最大值相当于求的最小值，所以上述目标函数等价为：

 因为现在的目标函数是二次的，约束条件是线性的，所以它是一个凸二次规划问题。

### 2.4 拉格朗日乘子法

由于这个问题的特殊结构，可以通过拉格朗日对偶性变换到对偶变量 的优化问题，即通过求解与原问题等价的对偶问题得到原始问题的最优解，这就是线性可分条件下支持向量机的对偶算法，这样做的优点在于：一者对偶问题往往更容易求解；二者可以自然的引入核函数，进而推广到非线性分类问题。

拉格朗日乘子法的具体步骤就是对于每一个约束（不等式约束g(x)、等式约束h(x)）引入一个拉格朗日乘子（），本问题中只有不等式约束，因此可将拉格朗日函数写作：

令

由上可知，当某个约束条件不满足时，即存在，那么显然有（因为可以是任意大于等于0的数，此处令）。而当所有约束条件都满足时，则最优值为，也就是目标函数中要最小化的值。因此最小化就等价于最小化。原目标函数就可以改写为：

其中，表示上述问题的最优值，且和最初的问题是等价的。但是直接求解这个问题又过于麻烦，因为先是和两个参数，而后又是不等式约束。因此，不妨把和的位置交换一下，将上述问题转化为对偶问题：

根据对偶问题的弱对偶性可知，。并由推论知，时，分别为原问题和对偶问题的最优解。

1. 求

对拉格朗日函数分别对求偏导数并令其等于0。

得

将其代入拉格朗日函数，既得

即

1. 求

将目标函数转换为极小

由此可求得对偶问题的最优解，根据对偶原理，可由求得原问题的最优解

### 2.5 利用KKT条件求解

该最优化问题满足KKT (Karush-Kuhn-Tucker) 条件，即得

由此得

故分离超平面可以写作

### 2.6 支持向量

最后阐述一下支持向量的含义。所有坐落在间隔边界上的点被称作是 “支持向量”，如图3(b)中用红色、蓝色小圆圈出来的点。

由KKT条件知，

对于的样本，有

即一定间隔边界上，即为支持向量。

## 三、解题过程

本文使用python语言对所提供的数据集进行数据清洗，并利用sklearn中自带的SVM函数对该问题进行求解。在本节后半部分会详述SVM函数的内部结构，并独立完成SVM函数的编写以展示此类凸二次规划的解题方法。

### 3.1 数据标签

题目中提供的训练数据集train.csv含有891个乘客信息，每位乘客有12个属性，数据集截图如图4：

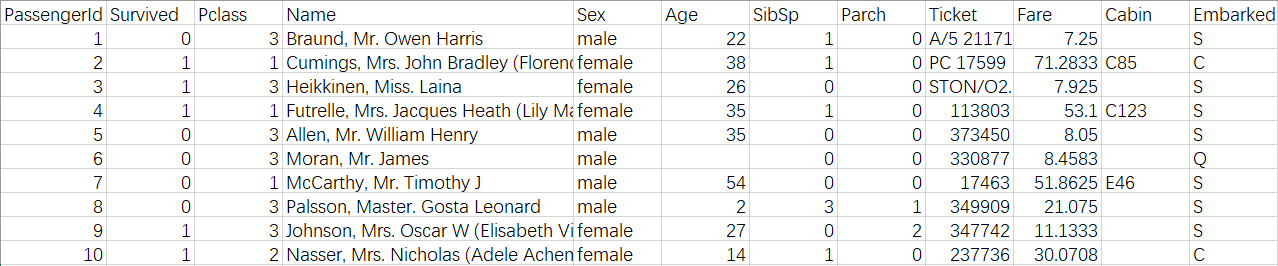


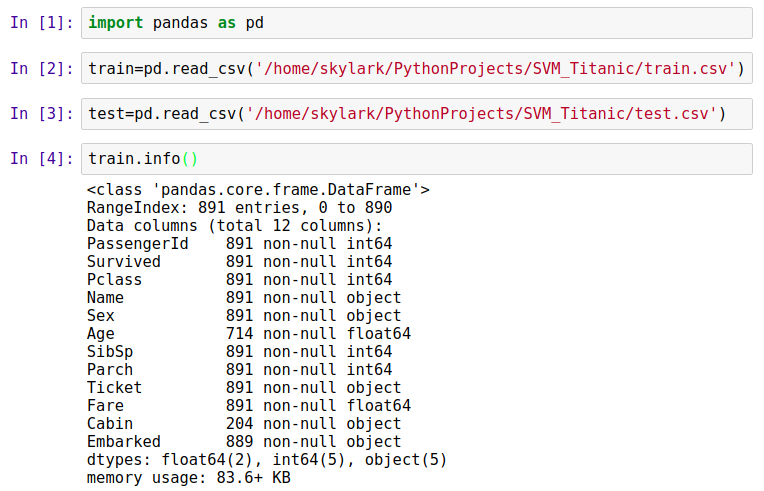
图4 数据集截图

除了Survived（表示是否获救）外，其他是乘客的信息：

* PassengerId ：乘客ID
* Pclass ：乘客等级(1/2/3等舱位)
* Name ： 乘客姓名
* Sex ： 性别
* Age ： 年龄
* SibSp ： 堂兄弟/妹个数
* Parch ： 父母与小孩个数
* Ticket ： 船票信息
* Fare ：票价
* Cabin ： 客舱
* Embarked ： 登船港口

### 3.2 数据清洗

利用Pandas库，导入csv数据，我们可以了解数据集的基本情况。



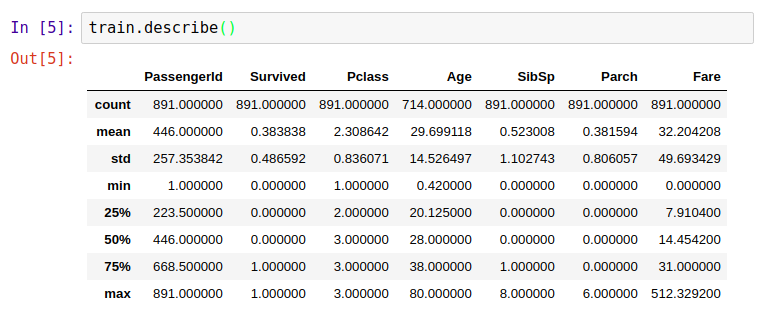
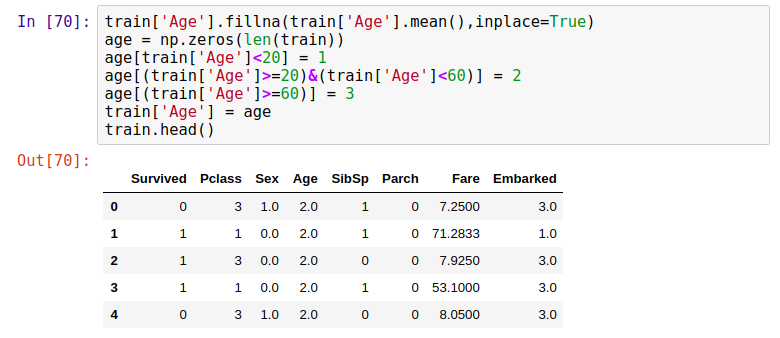
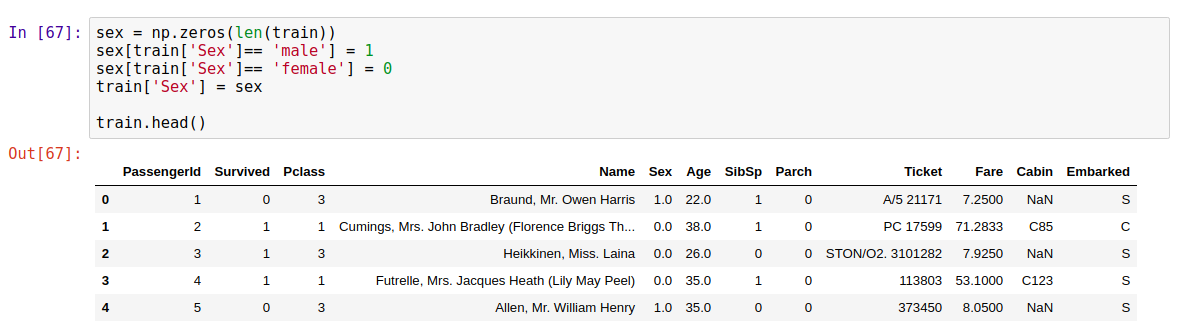


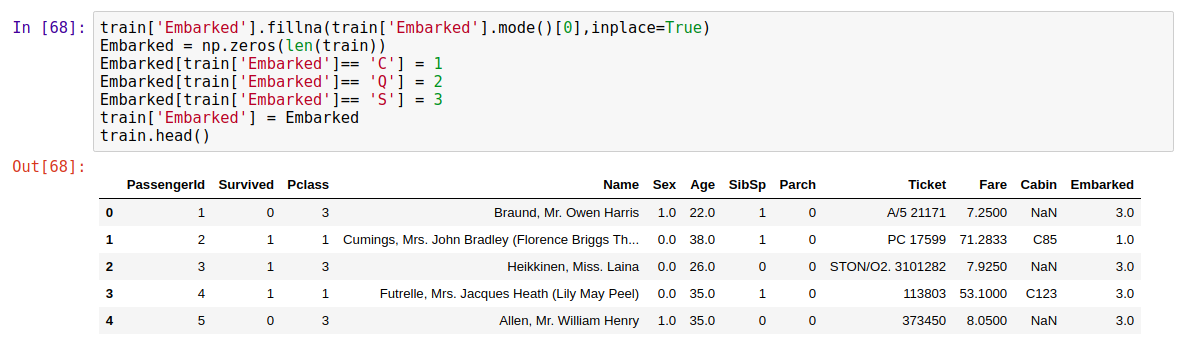
图5 数据集基本情况

由图5可知，Age和Cabin特征出现了数据缺失的情况，其中Cabin特征缺失较为严重，只有204条记录，可以考虑舍弃该特征。而Age特征可以通过fillna()函数填补缺失值, 填充值为Age的平均值，并将其分为（1、2、3）三个等级，方便数据处理：

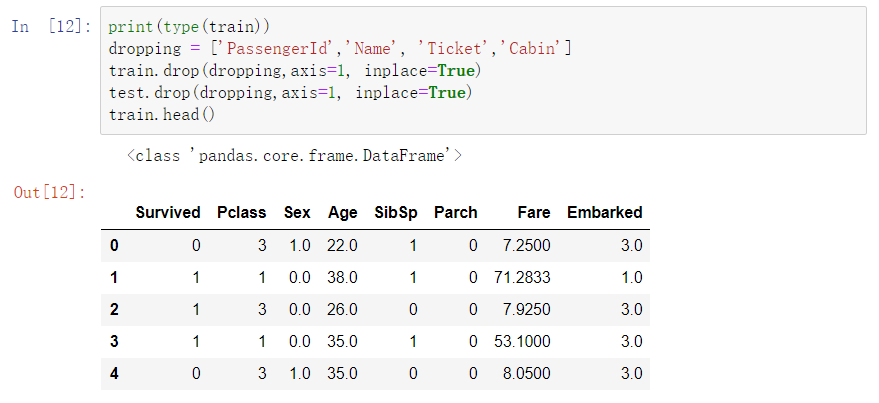


此外，还需要对几个非数值类型的数据进行替换，包括Sex、Embarked两类特征。





删除数据集中的无用信息PassengerId、Name、Ticket、Cabin特征：



### 3.3 数据分析

计算特征之间的相关性，并绘制热力图：

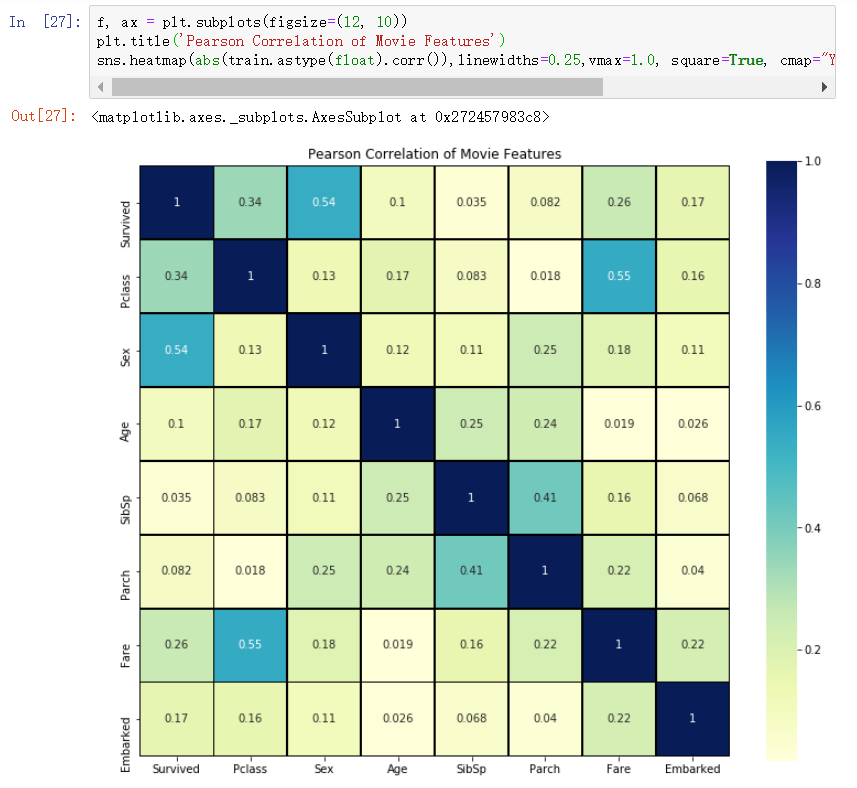


图4 特征相关性热力图

上图中颜色越深表示特征之间的相关性越大，故Sex、Pclass、Fare对能否存活(Survived)影响较大。

根据Sex和Pclass特征，可以绘制出如下的关系图：

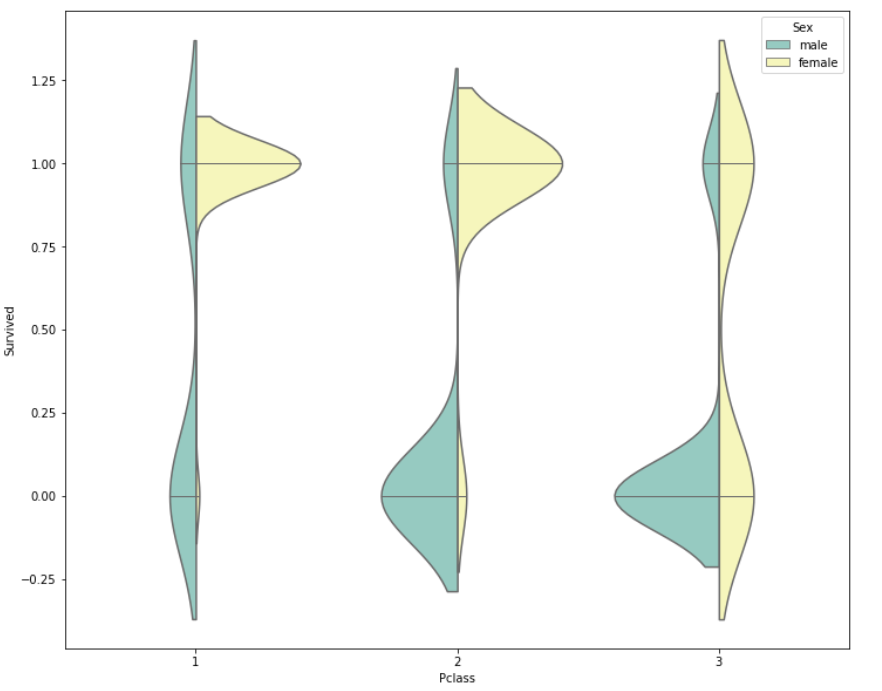
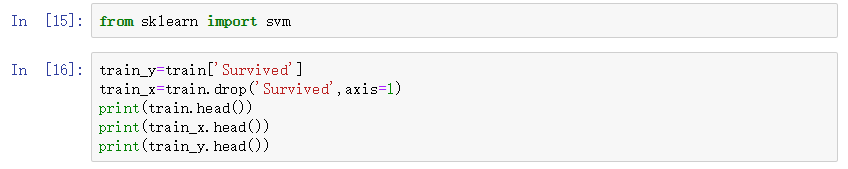


图5 Sex和Pclass特征与Survived关系的Violin图

上图直观地揭示了，有钱人以及女性更容易在泰坦尼克事故中幸存下来，而穷人以及男性等特征会增加其在事故中的死亡率。

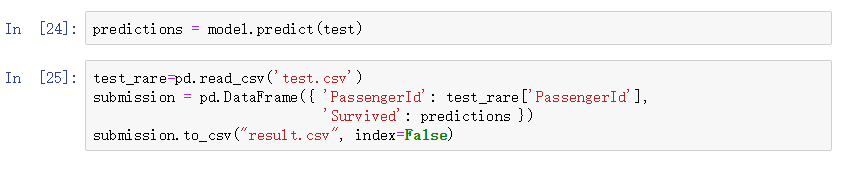
### 3.4 建立SVM模型并预测

使用sklearn库中的svm函数，并将处理过的训练数据集分为Survived和其他两部分。

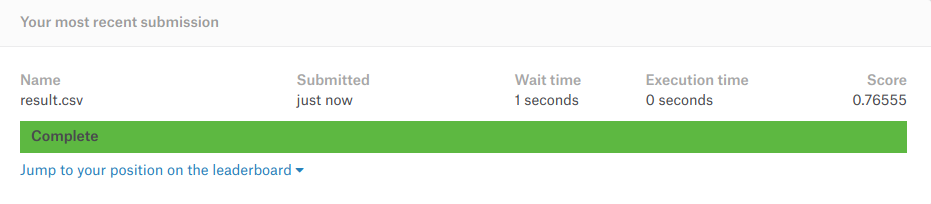


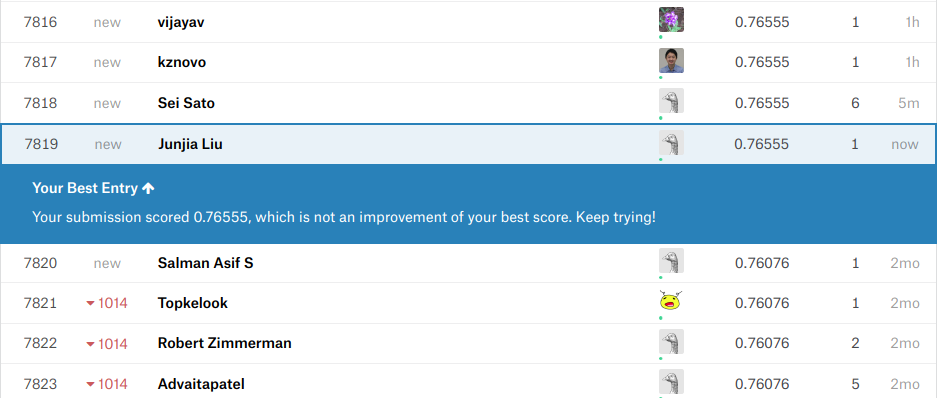


此处用到了线性核，用于最基本的线性可分情况。对于核函数的选用，本文就不过多涉及。至此，我们已经训练出了分类模型，将其应用到测试数据集的预测中，注意测试数据集也需要相同方式的数据处理，此处不再赘述，详见附件程序。预测过程如下：



预测结果被存入result.csv文件中，将其提交到Kaggle上看一下正确率：





很遗憾，得分0.76555，排名7819名。由此可见，SVM对于这道题来说可能并不是个好办法，但这就不是本文的重点了

### 3.5 SVM的python实现

本程序省略了数据处理部分，直接使用了上面IPython程序处理后的train\_new.csv和test\_new.csv数据集。由于原题给出的测试数据集(test.csv)实际上是预测数据集，缺少标签。因此在本程序的将原训练数据集按传统方法分为7:3两部分，30%的训练数据不参加训练而是应用到测试环节，用于观察模型是否过拟合。具体思路见下面代码及注释 ：

1. # -\*- coding: utf-8 -\*-
2. """
3. Pycharm
5. by Junjia Liu
7. """
9. **from** numpy **import** \*
10. **import** pandas as pd
12. **def** loadDataSet(filename): #读取数据
14. f=pd.read\_csv(filename)
15. # train=f[ :630, :]
16. # test=f[631: , :]
17. labelMat=f['Survived']
18. labelMat.replace(0, -1, inplace=True)
19. dataMat=f.drop('Survived',axis=1)
20. **return** dataMat,labelMat #返回数据特征和数据类别
22. **def** selectJrand(i,m): #在0-m中随机选择一个不是i的整数
23. j=i
24. **while** (j==i):
25. j=int(random.uniform(0,m))
26. **return** j
28. #最优解剪辑，《统计学习方法》p127 7.108
29. **def** clipAlpha(aj,H,L):  #保证a在L和H范围内（L <= a <= H）
30. **if** aj>H:
31. aj=H
32. **if** L>aj:
33. aj=L
34. **return** aj
36. **def** kernelTrans(X, A, kTup): #核函数，输入参数,X:支持向量的特征树；A：某一行特征数据；kTup：('lin',k1)核函数的类型和参数
37. m,n = shape(X)
38. K = mat(zeros((m,1)))
39. **if** kTup[0]=='lin': #线性函数
40. K = X \* A.T
41. **elif** kTup[0]=='rbf': # 径向基函数(radial bias function)
42. **for** j **in** range(m):
43. deltaRow = X[j,:] - A
44. K[j] = deltaRow\*deltaRow.T
45. K = exp(K/(-1\*kTup[1]\*\*2)) #返回生成的结果
46. **else**:
47. **raise** NameError('Houston We Have a Problem -- That Kernel is not recognized')
48. **return** K

51. #定义类，方便存储数据
52. **class** optStruct:
53. **def** \_\_init\_\_(self, dataMatIn, classLabels, C, toler, kTup):  # 存储各类参数
54. self.X = dataMatIn  #数据特征
55. self.labelMat = classLabels #数据类别
56. self.C = C #软间隔参数C，参数越大，非线性拟合能力越强
57. self.tol = toler #停止阀值
58. self.m = shape(dataMatIn)[0] #数据行数
59. self.alphas = mat(zeros((self.m,1)))
60. self.b = 0 #初始设为0
61. self.eCache = mat(zeros((self.m,2))) #缓存
62. self.K = mat(zeros((self.m,self.m))) #核函数的计算结果
63. **for** i **in** range(self.m):
64. self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

67. **def** calcEk(oS, k): #计算Ek（参考《统计学习方法》p127公式7.105）
68. gXk = float(multiply(oS.alphas,oS.labelMat).T\*oS.K[:,k] + oS.b) #7.104
69. # print(float(multiply(oS.alphas,oS.labelMat).T\*oS.K[:,k]))
70. Ek = gXk - float(oS.labelMat[k])
71. **return** Ek
73. #随机选取aj，并返回其E值
74. **def** selectJ(i, oS, Ei):
75. maxK = -1
76. maxDeltaE = 0
77. Ej = 0
78. oS.eCache[i] = [1,Ei]
79. validEcacheList = nonzero(oS.eCache[:,0].A)[0]  #返回矩阵中的非零位置的行数
80. **if** (len(validEcacheList)) > 1:
81. **for** k **in** validEcacheList:
82. **if** k == i:
83. **continue**
84. Ek = calcEk(oS, k)
85. deltaE = abs(Ei - Ek)
86. **if** (deltaE > maxDeltaE): #返回步长最大的aj
87. maxK = k
88. maxDeltaE = deltaE
89. Ej = Ek
90. **return** maxK, Ej
91. **else**:
92. j = selectJrand(i, oS.m)
93. Ej = calcEk(oS, j)
94. **return** j, Ej

97. **def** updateEk(oS, k): #更新os数据
98. Ek = calcEk(oS, k)
99. oS.eCache[k] = [1,Ek]
101. #首先检验ai是否满足KKT条件，如果不满足，随机选择aj进行优化，更新ai,aj,b值
102. **def** innerL(i, oS): #输入参数i和所有参数数据
103. Ei = calcEk(oS, i) #计算E值
104. # print(float(oS.labelMat[i]\*Ei)+oS.alphas[i])
105. **if** ((oS.labelMat[i]\*Ei < -oS.tol) **and** (oS.alphas[i] < oS.C)) **or** ((oS.labelMat[i]\*Ei > oS.tol) **and** (oS.alphas[i] > 0)): #检验这行数据是否符合KKT条件 参考《统计学习方法》p128公式7.111-113
106. j,Ej = selectJ(i, oS, Ei) #随机选取aj，并返回其E值
107. alphaIold = oS.alphas[i].copy()
108. alphaJold = oS.alphas[j].copy()
109. **if** (oS.labelMat[i] != oS.labelMat[j]): #以下代码的公式参考《统计学习方法》p126下侧，L、H取值计算
110. L = max(0, oS.alphas[j] - oS.alphas[i])
111. H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
112. **else**:
113. L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
114. H = min(oS.C, oS.alphas[j] + oS.alphas[i])
115. **if** L==H:
116. # print("Error: L==H")
117. **return** 0
118. eta = 2.0 \* oS.K[i,j] - oS.K[i,i] - oS.K[j,j]   #参考《统计学习方法》p127公式7.107
119. **if** eta >= 0:
120. # print("Error: eta>=0")
121. **return** 0
122. oS.alphas[j] -= oS.labelMat[j]\*(Ei - Ej)/eta #参考《统计学习方法》p127公式7.106，未经剪辑的情况下沿着约束方向更新alpha[j]
123. oS.alphas[j] = clipAlpha(oS.alphas[j],H,L) #参考《统计学习方法》p127公式7.108，对新alpha[j]进行剪辑
124. updateEk(oS, j)
125. **if** (abs(oS.alphas[j] - alphaJold) < oS.tol): #alpha变化大小阀值（自己设定）
126. # print("Error: j not moving enough")
127. **return** 0
128. oS.alphas[i] += oS.labelMat[j]\*oS.labelMat[i]\*(alphaJold - oS.alphas[j])#参考《统计学习方法》p127公式7.109， 由alpha[j]求得alpha[i]
129. updateEk(oS, i) #更新数据
130. #以下求解b的过程，参考《统计学习方法》p129公式7.115-7.116
131. b1 = oS.b - Ei- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,i] - oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[i,j]
132. b2 = oS.b - Ej- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,j]- oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[j,j]
133. **if** (0 < oS.alphas[i]<oS.C):
134. oS.b = b1
135. **elif** (0 < oS.alphas[j]<oS.C):
136. oS.b = b2
137. **else**:
138. oS.b = (b1 + b2)/2.0
139. **return** 1
140. **else**:
141. # print("Error:")
142. **return** 0

145. #SMO函数，用于快速求解出alpha
146. **def** smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)): #输入参数：数据特征，数据类别，参数C，阀值toler，最大迭代次数，核函数（默认线性核）
147. oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler, kTup)
148. iter = 0
149. entireSet = True
150. alphaPairsChanged = 0
151. **while** (iter < maxIter) **and** ((alphaPairsChanged > 0) **or** (entireSet)):
152. alphaPairsChanged = 0
153. **if** entireSet:
154. **for** i **in** range(oS.m): #遍历所有数据
155. alphaPairsChanged += innerL(i,oS)
156. #print("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)) #显示第多少次迭代，那行特征数据使alpha发生了改变，这次改变了多少次alpha
157. iter += 1
158. **else**:
159. nonBoundIs = nonzero((oS.alphas.A > 0) \* (oS.alphas.A < C))[0]
160. **for** i **in** nonBoundIs: #遍历非边界的数据
161. alphaPairsChanged += innerL(i,oS)
162. # print("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
163. iter += 1
164. **if** entireSet:
165. entireSet = False
166. **elif** (alphaPairsChanged == 0):
167. entireSet = True
168. **print**("iteration number: %d" % iter)
169. **return** oS.b,oS.alphas
171. **def** testRbf(data\_train, data\_predict):
172. dataArr,labelArr = loadDataSet(data\_train) #读取训练数据
174. #因为题目中提供的test数据是没有标签的，所以测试集就按照3:7的比例从训练数据集中分出
175. dataArr\_train=dataArr.values[ :630, :]
176. labelArr\_train=labelArr.values[ :630]
177. dataArr\_test=dataArr.values[631: , :]
178. labelArr\_test=labelArr.values[631: ]
180. b,alphas = smoP(dataArr\_train, labelArr\_train, 20, 0.0001, 12, ('rbf', 0.2)) #通过SMO算法得到b和alpha
181. datMat=mat(dataArr\_train)
182. labelMat = mat(labelArr\_train).transpose()
183. svInd=nonzero(alphas)[0]  #选取不为0数据的行数（也就是支持向量）
184. sVs=datMat[svInd] #支持向量的特征数据
185. labelSV = labelMat[svInd] #支持向量的类别（1或-1）
186. **print**("there are %d Support Vectors" % shape(sVs)[0]) #打印出共有多少的支持向量
187. m,n = shape(datMat) #训练数据的行列数
188. errorCount = 0
189. **for** i **in** range(m):
190. kernelEval = kernelTrans(sVs,datMat[i,:], ('rbf', 0.2)) #将支持向量转化为核函数
191. predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b  #这一行的预测结果（代码来源于《统计学习方法》p133里面最后用于预测的公式）注意最后确定的分离平面只有那些支持向量决定。
192. # print(sign(predict), sign(labelArr\_train[i]))
193. **if** sign(predict) != sign(labelArr\_train[i]): #sign函数 -1 if x < 0, 0 if x==0, 1 if x > 0
194. errorCount += 1
195. **print**("the training error rate is: %f" % (float(errorCount)/m)) #打印出错误率

198. errorCount\_test = 0
199. datMat\_test=mat(dataArr\_test)
200. labelMat\_test = mat(labelArr\_test).transpose()
201. m,n = shape(datMat\_test)
202. **for** i **in** range(m): #在测试数据上检验错误率
203. kernelEval = kernelTrans(sVs,datMat\_test[i,:], ('rbf', 0.2))
204. predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b
205. **if** sign(predict) != sign(labelArr\_test[i]):
206. errorCount\_test += 1
207. **print**("the test error rate is: %f" % (float(errorCount\_test)/m))
209. #Prediction
210. datMat\_predict=mat(data\_predict)
211. m,n=shape(datMat\_predict)
212. predictions=zeros(m)
213. **for** i **in** range(m):
214. kernelEval = kernelTrans(sVs, datMat\_predict[i, :], ('rbf', 0.2))
215. predictions[i] = kernelEval.T \* multiply(labelSV, alphas[svInd]) + b
216. avg\_predict=average(predictions)
217. **for** i **in** range(m):
218. **if** predictions[i] >= avg\_predict:
219. predictions[i] = 1
220. **else**:
221. predictions[i] = 0
222. predictions.astype('int')
223. test\_rare = pd.read\_csv('test.csv')
224. submission = pd.DataFrame({'PassengerId': test\_rare['PassengerId'],
225. 'Survived': predictions}).astype('int')
226. submission.to\_csv("myresult.csv", index=False)
228. #主程序, 直接应用IPython程序处理过的数据集train\_new和test\_new
229. **def** main():
230. filename\_traindata='train\_new.csv'
231. filename\_predictdata=pd.read\_csv('test\_new.csv')
232. testRbf(filename\_traindata, filename\_predictdata)
234. **if** \_\_name\_\_=='\_\_main\_\_':
235. main()

经过多次调参，在使用高斯核函数、软间隔系数且误差的情况下，得到了较为理想的模型。应用该模型预测test\_new.csv数据集，并将结果处理后，提交到Kaggle，结果如下：

图片包含 屏幕截图

自动生成的说明

这一次的成绩为0.66028，较上次还有所下降。但重要的是，本程序基于支持向量机的原理实现了与sklearn库中svm函数等同的功能，也是本文创新所在。

# 参考文献

[1] 李航. 统计学习方法[J]. 2012.